```java
/*
 * Copyright (c) 1995 Luis Fernandes
 *
 * Permission to use, copy, hack, and distribute this software and its
 * documentation for NON-COMMERCIAL purposes without fee is hereby
 * granted provided that this copyright notice appears in all copies.
 *
 * COMMERCIAL licenses are available for a tiny cost.
 *
 */

/* $Id: Abacus.java,v 1.7 1996/05/30 20:22:16 elf Exp $ */

/* $Log: Abacus.java,v $
 * Revision 1.7  1996/05/30 20:22:16  elf
 * Support for parameters to override images for the frame and beads.
 *
 * Support for tourist information (customizable params, etc.) via the
 * appletviewer-menu added.
 *
 * Removed ux,uy (clipping coords) which were not used.
 *
 * Image-update is bouble-buffered (done in the previous version).
 *
 * After reading JAG/KA's _TJPL_, I've decided that this applet needs a
 * major re-write.
 *
 * Revision 1.6  1996/05/22 19:33:33  elf
 * The frame is now a gif representation.
 *
 * Revision 1.5  1996/04/04  18:32:05  elf
 * Also display the value in the java console, since there is a bug in
 * Netscape that does not show it in the frame.
 *
 * Revision 1.4  1996/03/01  21:41:34  elf
 * In update(), instead of calling paint(), just re-draw the columns. The
 * re-draw looks better now.
 *
 * Revision 1.3  1996/01/12  19:18:40  elf
 * In mouseMove, if the column-returned in cx is >=NCOLS (which is
 * possible), trap-it (used to be an ArrayIndexOutOfBoundsException).
 * I think translateXY2RowCol should do this trapping.
 *
 * Revision 1.2  1996/01/10  20:47:29  elf
 * Track the mouse and display descriptive-information in the status
 * area about the anatomy of the abacus.
 *
 * Revision 1.1  1995/09/25  20:15:07  elf
 * Initial revision
 */

import java.awt.*;
```

```java
/** This is a java-applet simulation of a Chinese abacus.
 *
 *
 * Each column is stored internally in an integer array called
 * 'column' that is initialized to represent the resting positions of
 * the beads: 2 bead on top deck row-pos 0 & 1, pos 2 empty; 5 beads
 * lower deck pos 4-8, pos 3 empty. The initial value is
 * 499d=111110011 where a '1' represents a position that is occupied
 * by a bead and a '0' represents an empty position. As the beads are
 * moved, the 1's are shifted (using << and >>) accordingly to
 * represent their new locations.
 *
 * The illustration below represents a single column of the abacus.
 *
<pre>
   O represents the bead;
   | represents an empty position (the rod)
   = represents the frame

        =============    Row Position (index 'r' )
              O                     0
              O                     1                   Upper Deck
              |                     2
        =============
              |                     3
              O                     4
              O                     5                   Lower Deck
              O                     6
              O                     7
              O                     8
        =============
</pre>
 *
 * @author Luis Fernandes
 */

public class Abacus extends java.applet.Applet
{
   /*  initial attributes of the abacus*/
  static final int MAXCOLS=10;   /* number of columns the abacus will have */

  static final int BEADHEIGHT=17;
  static final int BEADWIDTH=17;
  static final int FRAMEWIDTH=10;  /* thickness of the frame*/

  static final int COLGAP=2;       /* gap between 2 cols*/
  static final int ROWGAP=2;       /* gap between 2 rows*/

  static final int NTOPROWS=3;   /*(2 beads, 1 gap on top-deck)*/
  static final int NBOTROWS=6;   /*(5 beads, 1 gap on top-deck)*/

  static final int NCOLS=MAXCOLS;  /* number of columns*/
```

```java
static final int NROWS=(NTOPROWS+NBOTROWS);  /* number of rows*/

static final int MIDFRAME=(FRAMEWIDTH+(NTOPROWS*BEADHEIGHT)+((NTOPROWS+1)*ROWGAP));

//size().width & size().height of window depends on attributes of the abacus
static final int INITWIDTH=((2*FRAMEWIDTH)+(NCOLS*BEADWIDTH)+((NCOLS+1)*COLGAP));
static final int INITHEIGHT=((3*FRAMEWIDTH)+(NROWS*BEADHEIGHT)+((NROWS+1)*ROWGAP));

private int column[]=new int[MAXCOLS];

private int cx, cy;                    // for xlating x,y to row,col

private boolean overflow;              // when the abacus overflows this is true
private boolean carry;                 // when a calc causes a carry to the next col

Image bead, nobead;                    // picture of a bead and an empty loc
Image frame;                                   // picture of the frame

String [] defaultImageName=
        {
          "images/frame.gif", "images/diamond.gif", "images/nodiamond.gif",
        };


Image buf;                                     // off-screen image
Graphics gbuf;                                 // offscreen GC

Font valueFont;                                // font used to paint the value

    /**
     * Tourist information. (Applet->Info)
     */
public String getAppletInfo()
        {
          String ver= "$Id: Abacus.java,v 1.7 1996/05/30 20:22:16 elf Exp $";

          return "Chinese Abacus by Luis Fernandes\n"+ver;
        }

    /**
     * Parameters user can customize.(Applet->Info)
     */
   public String[][] getParameterInfo()
          {
                String[][] info=
                  {
                        {"value",     "integer",  "initial value of the abacus (max. "+MAXCOLS+" digits)"},
                        {"frameImg",  "gif URL",  "image for the frame (\""+defaultImageName[0]+"\" "+BEADHEIGHT+"X"+BEADWIDTH+" pixels)"},
                        {"beadImg",   "gif URL",  "image for a bead (\""+defaultImageName[1]+"\" "+BEADHEIGHT+"X"+BEADWIDTH+" pixels)"},
                        {"nobeadImg", "gif URL",  "image for an empty location (\""+defaultImageName[2]+"\" "+BEADHEIGHT+"X"+BEADWIDTH+" pixels)"},

                  };
                return info;
```

```
    }

  /**
        Initialize the Abacus by initializing the internal column
        array. If a "VALUE" resource is specified, set the column array
        according to it, instead.
        */

  public
        void init()
          {

                /* Init the internal configuration of the beads: 499d=111110011
                 * (2 bead on top deck pos 0 & 1; 5 beads lower deck pos 4-8,
                 * pos 2 & 3 are empty initially) */
                for(int i=0; i<MAXCOLS; i++) column[i]=499;

                String param=getParameter( "value" );
/*               System.out.println("==>"+param); */

                /* check for user-specified value resource*/
                if((param==null) || (param.length()>MAXCOLS) )
                  {
                        /* if no attribute is specified, or the value is too big,
                         * use default*/
                        System.out.println( "Ignoring VALUE resource: " +param+ "\n" );

                  }
                else            /* set each column according to the user-specified value*/
                  {
                        int len=param.length();

                        for(int i=0; i<len; i++)
                          {
                                int val=Integer.valueOf(param.substring(i,i+1)).intValue();

                                // set value in the upper-deck
                                if(val>4)
                                  {
/*                                      System.out.println("Col "+i+"%5="+val%5); */

                                        animateBead(1, i);
                                  }

                                // set value in the lower-deck
                                int remainder=val%5;

                                if(remainder>0)
                                  {
                                        animateBead(3+remainder, i);
                                  }
                          }
                  }
```

```
            param=getParameter( "frameImg" );

            if(param == null)
              {
                        frame=getImage(getCodeBase(), defaultImageName[0]);
              }
            else
              {
                        frame=getImage(getCodeBase(), param);
              }

            param=getParameter( "beadImg" );

            if(param == null)
              {
                        bead=getImage(getCodeBase(), defaultImageName[1]);
              }
            else
              {
                        bead=getImage(getCodeBase(), param);
              }

            param=getParameter( "nobeadImg" );
            if(param == null)
              {
                        nobead=getImage(getCodeBase(), defaultImageName[2]);
              }
            else
              {
                        nobead=getImage(getCodeBase(), param);
              }

            valueFont=new java.awt.Font( "Times" , Font.BOLD, 10);

            resize(INITWIDTH, INITHEIGHT);  /* initial size */

            buf = createImage(INITWIDTH,INITHEIGHT);
            gbuf = buf.getGraphics();


      }

private
        void displayValue(Graphics g)
      {
            char valchars[]= new char [MAXCOLS*3];

            String value= new String (valchars);

            if(overflow) value+= "*" ;

             /* look at each column*/
```

```
		for(int col=0; col<MAXCOLS; col++)
		  {
			int r;
			int val=0;

			/* find the empty row, and determine what the value is*/
			/* top-deck*/
			for(r=2; r>=0; r--)
			  {
				if(!RowOccupied(r, column[col])) break;
			  }

			val+=((2-r)*5);

			/* bottom-deck*/
			for(r=3; r<9; r++)
			  {
				if(!RowOccupied(r, column[col])) break;
			  }

			val+=(r-3);

			value=value+ " " +Integer.toString(val);

/*			System.out.println("\tColumn"+col+"value="+val);*/
			if(overflow)
			  g.setColor(Color.red);
			else
			  g.setColor(Color.black);

			g.drawString(Integer.toString(val),
						FRAMEWIDTH+COLGAP+(col*(COLGAP+BEADWIDTH)), 10);

		  }


		/* kludge for Netscape because some version do not display the value
		   in the frame*/
//		System.out.println("Value displayed is "+value);
	  }

  /**
	* Draw the abacus :draw the frame, draw the rods & draw the
	* beads; display the value in the frame.
	*/
  public
	void paint(Graphics g)
	  {
		if (bead == null) {
			return;
		}

		drawFrame(gbuf);
```

```
                for(int i=0; i<MAXCOLS; i++)
                        {
                          drawColumn(gbuf, i);
                        }

                displayValue(gbuf);

          g.drawImage(buf,0,0,this);  // copy off-screen buffer into applet

          }

public void update(Graphics g)
        {
          paint(g);
  }

 /**
        * Handle a mouse-click.
        *
        * Convert the x/y position of the click to a row/column equivalent
        * and move the corresponding bead.
        *
        * @param evt the internal AWT event varaible
        * @param x the x-position of the click, in pixels
        * @param y the y-position of the click, in pixels
        *
        */
public
        boolean mouseUp(java.awt.Event evt, int x, int y) {

           /* row,col returned in cx,cy*/
          boolean i=translateXY2RowCol(x, y);

          if(i)                                                /*valid row,col...*/
                {

                  if(RowOccupied(cy,column[cx]))
                        {
                          animateBead(cy,cx);

                          repaint();
                        }
                }
          return true;
        }  /*mouseUp()*/

 /**
        * Track the mouse.
        *
        * Display help information in the status area as the mouse moves
        * over various hot-points.
        *
```

```
         * @param evt the internal AWT event varaible
         * @param x the x-position of the mouse, in pixels
         * @param y the y-position of the mouse, in pixels
         */

  public boolean mouseMove(java.awt.Event evt, int x, int y)
         {

             /*        System.out.println("mouseMove(" + x + "," + y + ")");*/

            if(y>INITHEIGHT-FRAMEWIDTH || y<FRAMEWIDTH)
                  {
                    showStatus( "Frame" ); return true;
                  }

            if(x>INITWIDTH-FRAMEWIDTH || x<FRAMEWIDTH)
                  {
                    showStatus( "Frame" ); return true;
                  }

            if((y>MIDFRAME) && (y<MIDFRAME+FRAMEWIDTH))
                  {
                    showStatus( "Middle Frame (or Beam)" ); return true;
                  }

            if(translateXY2RowCol(x, y))
                  {
                    if(cx>=NCOLS) cx=NCOLS-1;

                    if(RowOccupied(cy, column[cx]))
                           {
/*                           System.out.println("Row:"+cy+", Col:"+cx+" Occupied");*/

                              if(cy<3)
                                    {
                                      showStatus( "Bead, value=5 (upper deck (" +(cy+1)+ "," +(cx+1)+ "))" ); return true;
                                    }
                              else
                                    {
                                      showStatus( "Bead, value=1 (lower deck (" +(cy+1)+ "," +(cx+1)+ "))" ); return true;
                                    }

                           }
                    else
                          {
                             if(x==FRAMEWIDTH+COLGAP+(BEADWIDTH/2)+(cx*(BEADWIDTH+COLGAP)))
                                    {
                                      showStatus( "Rod (column " +(cx+1)+ ")" ); return true;
                                    }
                          }

                  }
```

```
        showStatus( "" );

        return true;
    }

  private
      void drawColumn(Graphics g, int col)
        {

            for(int beadnum=0; beadnum<9; beadnum++)
              {
                  if(RowOccupied(beadnum,column[col]))
                    {
                        drawBead(g, beadnum, col);
                    }
                  else
                    {
                        undrawBead(g, beadnum, col);
                    }
              }

        } /*drawColumn()*/

  private
      void drawBead(Graphics g, int row, int col)
        {

            if(row<3)                          /* beads in the top-deck */
              {
                  g.drawImage(bead, FRAMEWIDTH+COLGAP+(col*(BEADWIDTH+COLGAP)),
                                    FRAMEWIDTH+(row*(BEADHEIGHT+ROWGAP))+2, this);
              }
            else                               /* account for the middle-rail */
              {
                  g.drawImage(bead, FRAMEWIDTH+COLGAP+(col*(BEADWIDTH+COLGAP)),
                                    (2*FRAMEWIDTH)+(row*(BEADHEIGHT+ROWGAP))+2, this);
              }

        } /*drawBead()*/

  private
      void undrawBead(Graphics g, int row, int col)
        {

            if(row<3)                          /* beads in the top-deck */
              {
                  g.drawImage(nobead, FRAMEWIDTH+COLGAP+(col*(BEADWIDTH+COLGAP)),
                                      FRAMEWIDTH+(row*(BEADHEIGHT+ROWGAP))+2, this);
              }
            else                               /* account for the middle-rail */
              {
                  g.drawImage(nobead, FRAMEWIDTH+COLGAP+(col*(BEADWIDTH+COLGAP)),
                                      (2*FRAMEWIDTH)+(row*(BEADHEIGHT+ROWGAP))+2, this);
```

```
			}

		}  /*undrawBead()*/



	private
		void drawFrame(Graphics g)
		{
			g.drawImage(frame, 0,0, this);

		} /*drawFrame()*/

	/**
	 * Moves the bead specified by r & c in the appropriate direction
	 *
	 * @param r is the row number for the bead
	 * @param c is the column number for the bead
	 */
	public
		void animateBead(int r, int c)
		{

			if(r<3)                                   /* selection in upper deck*/
			{
				int i;

				/* find empty row in upperdeck; pos 0,1,2*/
				for(i=0; i<3; i++) if(!RowOccupied(i,column[c])) break;

				if(i<r)          /* if empty row is above cur bead...*/
					moveBeadUp(r,c);
				else
					moveBeadDn(r,c);
			}
			else                                      /* selection in lower deck*/
			{
				int i;

				/* find empty row in lowerdeck; pos 3-8 */
				for(i=3; i<9; i++) if(!RowOccupied(i,column[c])) break;

				if(i<r)       /* if empty row is above cur bead...*/
					moveBeadUp(r,c);
				else
					moveBeadDn(r,c);

			}


		} /*animateBead()*/

	private
```

```
        void moveBeadUp(int r, int c)
          {
                  /* keep looking for an empty position directly above*/
                 if(RowOccupied(r-1,column[c])) moveBeadUp(r-1,c);

                 column[c]=column[c]-(1<<r);  /* row 'r' is now empty */
                 column[c]=column[c]+(1<<((r-1)));  /* row 'r-1' now is occupied*/

                  // check whether we have to carry to next column
                 if(!RowOccupied(0,column[c]))
                    {
                           System.out.println( "Carry to col" +(c-1));
                           carry=true;
                    }
                 else if(c==0 && r==1)
                    {
                           overflow=false;
                    }

          }  /*moveBeadUp()*/

  private
        void moveBeadDn(int r, int c)
          {

                  /* keep looking for an empty position directly below*/
                 if(RowOccupied(r+1,column[c])) moveBeadDn(r+1,c);

                 column[c]=column[c]-(1<<r);  /* row 'r' is now empty */
                 column[c]=column[c]+(1<<((r+1)));  /* row 'r+1' now is occupied*/

                  // check whether we have to carry to next column if in the top frame
                 if(!RowOccupied(0,column[c]) && r<3)
                    {
                           if(c==0)
                              {
                                     System.out.println( "***OVERFLOW***" );
                                     overflow=true;
                                     return;
                              }


                           System.out.println( "Carry to col" +(c-1));
                           carry=true;
                    }


          } /* moveBeadDown()*/


         /** translateXY2RowCol, converts x,y coord a row,col coord. The top
          * deck has 3 rows (1 is empty), the bottom deck has 6 rows (1 is
          * empty). Row # 3 is invalid because the middle-frame occupies this
```

```
     * position. Coordinates for the bottom deck are adjusted for this
     * anomaly, i.e. row #4 on the screen, is actually index #3 into the
     * Column array.
     *
     * @param x absolute x coordinate representing the click−location
     * @param y absolute y coordinate representing the click−location
     * @return cx as a global, the row index (0−9)
     * @return cy as a global, the column index (0−NCOLS)
     * @return True if click was on a bead or empty position
     * @return False if click was at mid−frame location
     */

    private
      boolean translateXY2RowCol(int x, int y)
            {

              cx=(x−FRAMEWIDTH)/(COLGAP+BEADWIDTH);

              if(cy<MIDFRAME)
                    {
                      cy=(y−FRAMEWIDTH)/(ROWGAP+BEADHEIGHT);
                    }
              else   /* account for the middle−frame (+1 bead size().height) (+4 is fudge)*/
                    {
                      cy=(y−((2*FRAMEWIDTH)−BEADHEIGHT))/(ROWGAP+BEADHEIGHT);
                    }

               /* technically, the mid−frame occupies position 3 & click is invalid*/
              if(cy==3) return(false);

              else if(cy>2)                              /* if pos (row) is greater than 3 ...*/
                    {
                      cy−−;                                       /* ...adjust by 1 row*/
                      return(true);
                    }
              else
                    {
                      return(true);
                    }

            } /*translateXY2RowCol()*/


final private
      boolean RowOccupied(int r, int c)
        {

            if((c & 1<<(r))==0)
              {
                    return false;
              }
            else
              {
```

```
                return true;
            }

    } /*RowOccupied()*/

}
```